AMENDMENTS TO THE CLAIMS:

1. (Currently Amended) A method for modeling and simulating software running interactively directly or indirectly on at least one digital computer, comprising the steps of:
providing a display;
providing an input, wherein said input connects at least one of a pointing device, a keyboard and external interactive devices to said software;
providing an output, wherein said output connects a block of memory to said display;
providing a software controller, wherein said software controller is a programmable agent controlling said software to perform tasks;
providing a software modeling process, wherein said software modeling process models an interaction process between said software and said software controller, further comprising the sub-steps of:
(a) connecting said software controller with said software through an input and an output of said software,
(b) controlling said software by said software controller **automatically and programmatically**, and
(c) identifying a model of said software on-line**, wherein said model of said software includes input and output behavior of said software under control of said software controller; wherein said input behavior of said software includes at least one control enabling structure; wherein said control enabling structure interacts with said software controller; wherein said output behavior of said software includes at least one display screen region**; and
providing a software simulation process, wherein said software simulation process simulates said interaction process between said software and said software controller, further comprising the sub-steps of:
(d) connecting said software controller with said model of said software through a simulated input and a simulated output of said model of said software,
(e) controlling said model of said software by said software controller **automatically and programmatically**, and
(f) simulating said interaction process between said software and said software controller without said software presence; and
wherein said software simulation process is a new software that comprises said model of said software and said software controller.

2. (previously presented) The method of claim 1, further comprising the step of:

providing a discrete sampling domain, wherein said discrete sampling domain is a finite integer sequence K driven by said software controller with a current sampling k indicating the most recent sampling.

3. (Original) The method of claim 2, further comprising the steps of:
providing a software dynamic system to represent said software modeling process, wherein said software dynamic system is a discrete system defined over said discrete sampling domain K;
wherein said software simulation process simulates said software dynamic system.

4. (Original) The method of claim 3, further comprising the steps of:
factoring said software into an algorithm engine and a cursor engine;
identifying a model of said algorithm engine and a model of said cursor engine in said software modeling process;
controlling said model of said algorithm engine and said model of said cursor engine in parallel by said software controller in said software simulation process; and
combining an output of said model of said algorithm engine and an output of said model of said cursor engine as said simulated output of said model of said software in said software simulation process.

5. (Original) The method of claim 4, wherein said cursor engine is modeled, identified and simulated, wherein the method further comprises the steps of:
providing a cursor function, wherein said cursor function calculates a cursor output based on a cursor position, a data structure and a plurality of cursor images;
wherein said cursor function models said cursor engine analytically;
providing a cursor engine modeler, wherein said cursor engine modeler identifies said data structure and said plurality of cursor images while said cursor engine is controlled by said software controller in said software modeling process;
wherein said model of said cursor engine under control of said software controller comprises said cursor function with said identified data structure and said identified cursor images;
wherein said model of said cursor engine simulates said cursor engine in said software simulation process, further comprising the sub-steps:
controlling said model of said cursor engine by said software controller in said software simulation process;

calculating said cursor output by said cursor function in said software simulation process based on,

said identified data structure,

said identified cursor images, and

said cursor position that is supplied by said software controller;

whereby said cursor engine is simulated by said model of said cursor engine that is controlled by said software controller in said software simulation process without said software presence.

6. (Original) The method of claim 4, wherein said algorithm engine is modeled and identified, wherein the method further comprises the steps of:
providing an initial state of said algorithm engine, wherein said initial state of said algorithm engine is sampled at the first output of said software;
providing a update function, wherein said update function is a sequence of discrete updates over said discrete sampling domain;
wherein said update function with said initial state model said algorithm engine;
providing an algorithm engine modeler, wherein said algorithm engine modeler samples said sequence of discrete updates while said software is controlled by said software controller in said software modeling process;
wherein said model of said algorithm engine identified comprises said sampled initial state and said sampled sequence of discrete updates as a modeled algorithm engine.

7. (Original) The method of claim 6, wherein said algorithm engine modeler is
controlled by said software controller and said algorithm engine.

8. (Original) The method of claim 7, wherein said algorithm engine modeler is controlled by two sub-samplers, wherein the method further comprises the steps of:
providing a direct input and output sampler;
providing an internal computation sampler; and
wherein said direct input and output sampler and said internal computation sampler run mutually exclusively.

9. (Original) The method of claim 8, wherein said direct input and output sampler controls said algorithm engine modeler to sample at least one discrete update induced directly by a command from said software controller, wherein the method further comprises

the steps of:
issuing said command by said software controller;
waiting for T(α), wherein said α < 1;
calling said direct input and output sampler by said software controller;
triggering said algorithm engine modeler by said direct input and output sampler to sample said discrete update D(k+α)-D(k);
holding said command by said software controller;
advancing said current sampling k by one;
modeling D(k+α) as D(k+1).

10. (Original) The method of claim 9, wherein said direct input and output sampler is callable by said software controller programmatically.

11. (previously presented) The method of claim 8, wherein said internal computation
sampler controls said algorithm engine modeler to sample at least one discrete update, wherein the method further comprises the steps of:
(a) providing an internal computation flow, wherein said internal computation flow is executed by said algorithm engine;
(b) providing at least one anchored point in said internal computation flow, wherein said anchored point is an internal computation unit of said algorithm engine with its computation state transition to be sampled precisely;
(c) providing a software sensor, wherein said software sensor is attached into said anchor point;
(d) setting up said software sensor with a numberOfStates argument, wherein said numberOfStates argument specifies computation state transitions to be observed;
(e) driving said algorithm engine to said internal computation unit;
(f) calling said software sensor before said internal computation unit is processed;
(g) processing said internal computation unit;
(h) calling said software sensor after said internal computation unit is processed;
(i) subtracting said numberOfStates argument by one;
(j) if said numberOfStates argument is not zero, going back to (e); otherwise
(k) removing said software sensor.

12. (previously presented) The method of claim 11, wherein said software sensor is an
Event Probe that may perform any programmed function

including triggering said sampling by said algorithm engine modeler, wherein the method further comprises the steps of:
providing an anchored event, wherein said anchored event is a window message;
wherein said anchored event processing is said anchored point;
modifying a callback function address of an internal window message processing;
driving said algorithm engine to said internal window message processing;
providing a dispatch unit, wherein said dispatch unit checks a current message with said anchored event, if a match is found, then,
calling a BeforeEvent Probe, wherein said BeforeEvent Probe is said Event Probe, and
processing said anchored event processing,
calling an AfterEvent Probe, wherein said AfterEvent Probe is said Event Probe, and
returning from said internal window message processing;
otherwise, processing an internal window message process.

13. (previously presented) The method of claim 11, wherein said software sensor is an API Probe that may perform any programmed function including triggering said sampling by said algorithm engine modeler, wherein the method further comprises the steps of:
(a) providing a marker function, wherein said marker function is an application programming call that is modeled as said anchored point within said internal computation flow;
(b) modifying a binary execution image of said software to reroute a calling to said marker function to a new function, wherein said new function includes at least one of said API probes and a call to said marker function;
(c) driving said algorithm engine to said internal computation flow;
(d) proceeding to said marker function;
(e) calling a BeforeMarker probe before said marker function is processed, wherein said BeforeMarker is said API Probe;
(f) processing said marker function;
(g) calling an AfterMarker probe after said marker function is processed, wherein said AfterMarker is said API Probe;
(h) subtracting said numberOfStates argument by one;
(i) if said numberOfStates argument is not zero, going back to (c); otherwise
(k) controlling said algorithm engine to stop said processing of said internal computation flow;

(l) removing said BeforeMarker and AfterMarker probes.

14. (Original) The method of claim 11, wherein said step of providing at least
one anchored point in said internal computation flow is a part of
an analytical modeling of said algorithm engine.

15. (Original) The method of claim 11, further comprising the step of:
providing a controlled target, wherein said controlled target is
said software in said software modeling process, or said model of
said software in said software simulation process;
providing an input of said controlled target, wherein said input of
said controlled target is said input of said software in said
software modeling process, or said simulated input of said model
of said software in said software simulation process;
providing an output of said controlled target, wherein said output
of said controlled target is said output of said software in said
software modeling process, or said simulated output of said
model of said software in said software simulation process.

16. (Original) The method of claim 15, wherein said software controller
further comprises,
a work-flow controller, wherein said work-flow controller controls
said controlled target to manipulate at least one user interface;
and
a work-space controller, wherein said work-space controller
controls said controlled target to perform at least one design
function.

17. (Original) The method of claim 16, wherein said work-space controller
comprises at least one of a coordinate controller and a functional controller.

18. (Original) The method of claim 17, wherein said coordinate controller
performs said design function, wherein the method further
comprises the step of:
providing a list of coordinates, wherein said list of coordinates are
used to drive said input of said controlled target.

19. (Original) The method of claim 17, wherein said functional controller
performs said design function, wherein the method further comprises the
steps of:
providing at least one modelable function, wherein said
modelable function is called with parameters to calculate
coordinates;

wherein said coordinates are used to drive said input of said controlled target.

20. (Original) The method of claim 19, further comprising the step of: providing at least one software actuator, wherein said software actuator connects said software controller to said controlled target.

21. (Original) The method of claim 20, wherein said software actuator models at least one graphical user interface element of said controlled target.

22. (Original) The method of claim 21, wherein said software actuator comprises a work-flow actuator interfacing to said work-flow controller of said software controller, wherein the method further comprises the steps of:
sending a high level command from said work-flow controller to said work-flow actuator, wherein said high level command comprises a tag and an input command;
calculating a valid region based on said tag by said work-flow actuator;
translating said command to at least one primitive action by said work-flow actuator;
if said command is for said pointing device, then
    computing a target position by said work-flow actuator based on said valid region,
    driving said input of said controlled target to said target position by said work-flow actuator, and
    applying said primitive action to said target by said work-flow actuator;
else,
    applying said primitive action to said valid region by said work-flow actuator.

23. (Original) The method of claim 22, wherein said step of calculating said valid region based on said tag further comprises the steps of:
if running in said software modeling process, then
    powering said valid region by said software,
    identifying said valid region based on said tag on-line,
    calculating said valid region, and
    saving said valid region with said tag as a part of said model of said software;
otherwise, running in said software simulation process,
    simulating said valid region by said model of said software,

retrieving said valid region based on said tag from said model of said software, and
calculating said valid region.

24. (Original) The method of claim 23, wherein said step of identifying said valid region based on said tag in said software modeling process further comprises the steps of:
providing a unique textual name, wherein said unique textual name is said tag;
providing a unique relation, wherein said unique relation is coded in a string and assigned as said unique textual name;
finding a unique handle from said software based on said tag, wherein said handle is uniquely located by said unique relation that is coded in said tag;
identifying said valid region based on said unique handle that is powered by said software.

25. (Original) The method of claim 24, wherein said unique relation is a parent-children-sibling relation that is defined uniquely by said graphical user interface element of said software.

26. (Original) The method of claim 21, wherein said software actuator comprises a work-space actuator interfacing to said work-space controller of said software controller, wherein the method further comprises the steps of:
(a) sending a high level command from said work-space controller to said work-space actuator;
(b) calculating a coordinate by said work-space actuator;
(c) translating said high level command to at least one primitive action by said work-space actuator;
(d) driving said input of said controlled target to said coordinate by said work-space actuator;
(e) applying said primitive action to said coordinate by said workspace actuator;
(f) going back to (b) if at least one coordinate remains.

27. (Original) The method of claim 26, wherein said step of calculating said coordinates further comprises the steps of:
providing a list of coordinates, wherein said list of coordinates is an array of elements with each element having a coordinate (x,y);
wherein said coordinate is retrieved sequentially from said array until the end of said array is reached.

28. (Original) The method of claim 26, wherein said step of calculating said coordinate further comprises the steps of:
providing at least one modelable function, wherein said modelable function calculates said coordinate based on at least one parameter that is supplied by said software controller;
wherein said modelable function implemented by said workspace actuator is a mathematical function that is scalable.

29. (Original) The method of claim 21, wherein said software actuator is a system actuator if a system standard graphical user interface element is modeled by said software actuator.

30. (Original) The method of claim 21, wherein said software actuator is a custom actuator if a one custom-built graphical user interface element is modeled by said software actuator.

31. (Original) The method of claim 29, wherein said system actuator is reusable universally for a plurality of said software modeling processes and said software simulation processes.

32. (Original) The method of claim 30, wherein said custom actuator is reusable for a plurality of said software controllers for said software modeling process and said software simulation process.

33. (Original) The method of claim 23, wherein said software actuator installs at least one software sensor.

34. (Original) The method of claim 23, further comprising the steps of:
providing a running context of said controlled target, wherein said running context is dynamically reconstructed by said software actuator in said software simulation process;
providing a structured input and output, wherein said structured input and output structuralizes said input and output of said controlled target based on said running context;
providing a feedback from said controlled target to said software controller through said structured output of said controlled target;
inferring a causal relationship from said running context.

35. (Original) The method of claim 20, further comprising the steps of:
modularizing said software modeling process and said software simulation process;
replacing said software with said model of said software that is identified in said software modeling process;
switching from said software modeling process to said software

simulation process;
preserving said connection between said software controller and said software actuator so that said software controller is kept invariant in said modeling process and said simulation process.

36. (Original) The method of claim 20, wherein said algorithm engine is simulated, wherein the method further comprises the steps of:
controlling said modeled algorithm engine by said software controller;
simulating said algorithm engine.

37. (Original) The method of claim 36, wherein said step of controlling said modeled algorithm engine in said software simulation process further comprises the steps of:
providing a Dynamic Update, wherein said Dynamic Update simulates said output of said algorithm engine;
reconstructing an output of said modeled algorithm engine, wherein said output of said modeled algorithm engine is updated by said Dynamic Update in sync with said software controller;
providing at least one simulated sampler, wherein said simulated sampler simulates said sampler used in said software modeling process;
controlling said Dynamic Update to update one output of said modeled algorithm engine by said simulated sampler if said sampler samples one discrete update of said algorithm engine in said software modeling process.

38. (Original) The method of claim 37, wherein said simulated sampler is controlled by said software controller through said software actuator.

39. (Original) The method of claim 38, wherein said simulated sampler is a simulated direct input and output sampler that symmetrically simulates said direct input and output sampler used in said software modeling process.

40. (Original) The method of claim 38, wherein said simulated sampler is a simulated internal computation sampler that symmetrically simulates said internal computation sampler used in said software modeling process.

41. (Original) The method of claim 37, wherein said Dynamic Update downloads and streams from an external source to update said output of said modeled algorithm engine.

42. (Original) The method of claim 41, wherein said external source is at least one local file.

43. (Original) The method of claim 41, wherein said external source is at least one Internet server.

44. (Original) The method of claim 39, wherein said software actuator drives said simulated direct input and output sampler, wherein the method further comprises the steps of:
modeling a sampling that is induced directly by said command from said software controller;
simulating said sampling by calling said simulated direct input and output sampler.

45. (Original) The method of claim 39, wherein said software actuator simulates said internal computation flow of said algorithm engine, wherein the method further comprises the steps of:
modeling a sampling event, wherein said sampling event samples after or before said internal computation unit is processed;
simulating said sampling event by calling said simulated internal computation sampler.

46. (Original) The method of claim 38, further comprising the step of:
creating a simulated discrete sampling domain from the execution flow of said software controller in said software simulation process, wherein said simulated discrete sampling domain is identical causally to said discrete sampling domain.

47. (Original) The method of claim 46, further comprising the steps of:
providing a compressing transformation on said simulated discrete sampling domain, wherein said compressing transformation reduces a time duration between two samplings in said software simulation process while causality is preserved;
providing a stretching transformation on said simulated discrete sampling domain, wherein said stretching transformation extends a time duration between two samplings in said software simulation process while causality is preserved.

48. (Original) The method of claim 3, wherein said software modeling process is a software modeling automation that runs autonomously.

49. (Original) The method of claim 48, wherein said software simulation process is a software simulation automation that runs

autonomously, wherein the method further comprises the step:
providing an output of said software simulation automation,
wherein said output of said software simulation automation is
said output of said model of said software that is manipulable.

50. (Original) The method of claim 49, wherein said software simulation automation is augmented with additional computation while said software dynamic system is preserved.

51. (Original) The method of claim 50, wherein the step of augmenting said software simulation automation further comprises the steps of:
providing an interaction input component H, wherein said interaction input component H engages a user to interact with said software simulation automation;
providing an index component G, wherein said index component G controls visibility of said output of said software simulation automation;
providing a programmable extension component E, wherein said programmable extension component E extends programmatically said software simulation automation with additional computational process.

52. (Original) The method of claim 51, wherein said interaction input component H is inserted between said output of said software controller and said input of said model of said software, wherein the method further comprises the steps of:
(a) updating said software simulation automation at a current sampling k;
(b) executing a programmed interaction by said software controller;
(c) updating a running context, wherein said running context includes a valid region, a pointing device status, a keyboard status, and a current command which are generated from said programmed interaction by said software controller;
(d) transferring execution of said software simulation automation to said H;
(e) waiting for an interaction input from said user;
(f) receiving said interaction input from said user;
(g) comparing said interaction input with said running context;
(h) if said interaction input is matched with said running context, then continuing said execution of said software simulation automation;
(i) otherwise, going back to (e).

53. (Original) The method of claim 51, wherein said interaction input component H is independent of said software simulation automation and said His applicable universally to any software simulation automation.

54. (Original) The method of claim 51, wherein said interaction input component H can be plugged or unplugged into said software simulation automation programmatically so that said software simulation automation can run interactively or automatically.

55. (Original) The method of claim 51, wherein said index component G indexes said software simulation automation selectively, wherein the method further comprises the steps of:
(a) providing at least one index set K_I, wherein said K_I is a subset of said discrete sampling domain K;
(b) updating said output of said software simulation automation in a memory that is invisible;
(c) providing a gated output-mapping function, wherein said gated output-mapping function controls copying from said memory to said display based on said index set K_I;
(d) running said software simulation automation;
(e) updating a current output of said software simulation automation at a current sampling k in said memory;
(f) if said current sampling $k \in K_I$, then,
copying said current output from said memory to said display by said gated output-mapping function,
continuing said simulation;
(g) otherwise,
advancing said current sampling k by one,
going back to (e) as fast as computationally possible.

56. (Original) The method of claim 55, further comprising the steps of:
providing first textual term, wherein said first textual term is associated with a sampling k_1;
providing second textual term, wherein said second textual term is associated with a sampling k_2;
providing at least one pair of descriptive textual terms for searching, wherein one in said descriptive textual terms is said first textual term, and said other in said descriptive textual terms is said second textual term;
conducting searches based on said descriptive textual terms to find two samplings k_1 and k_2;
running said software simulation automation;
updating said current output of said software simulation

automation in said memory;
if said current sampling k >= k1 and k < k2, then
copying said output of said software simulation automation from said memory to said display by said gated output mapping function;
selectively running at least one segment of simulation from k_1 to k_2-1 visually while computationally running said software simulation automation internally.

57. (Original) The method of claim 51, wherein said programmable extension component E extends additional computation into said output of said software simulation automation, wherein the method further comprises the steps of:
providing a new programmable function, wherein said new programmable function implements a sub-process;
updating a current output of said software simulation automation at said current sampling k;
extending said new sub-process with said current output as an initial state and a next output of said software simulation automation as a final state;
computing at least one sub-state with said new programmable function, wherein said sub-state in said sub-process updates over said output of said software simulation automation;
updating said next output of said software simulation automation at said current sampling k+1.

58. (Original) The method of claim 57, wherein said new programmable function is a Microsoft Internet Explorer Browser component, wherein the method further comprises the steps of:
providing a DHTML page, wherein said DHTML page is loaded by said Microsoft Internet Explorer Browser component;
providing a Document Object Model (DOM), wherein said Document Object Model is programmed in said DHTML page;
providing a URL link, wherein said URL link is encoded in said DHTML page;
providing a target region, wherein said target region is defined by said Document Object Model;
clicking or typing in said target region;
navigating to said URL link.

59. (Original) The method of claim 52, further comprising the steps of:
providing a controlled target, wherein said controlled target is said Microsoft Internet Explorer Browser component;
lively running said Internet Explorer Browser component in said

software simulation automation;
programming web browsing activities by said software controller;
engaging said user with said web browsing activities interactively or automatically.

60. (Original) The method of claim 51, further comprising the steps of:
providing a software amplification device, wherein said software amplification device comprises
    said software controller,
    said interaction input component H,
    said index component G, and
    said programmable extension component E;
wherein said software amplification device connects with said model of said software in a top loop;
feeding an augmented output of said model of said software through said software amplification device;
sensing said augmented output by a user as an input from said software amplification device;
engaging said user with said software amplification device in a bottom loop;
making an interaction movement by said user as an output to said software amplification device; and
interacting said model of said software by said user through said software amplification device.

61. (Original) The method of claim 60, further comprising the steps of:
(a) running a sub-process in said programmable extension component E to prepare said user to take a right interaction, wherein said sub-process implements at least one programmable function;
(b) executing a programmed interaction, wherein said programmed interaction is performed by said software controller;
(c) updating a running context, wherein said running context includes a valid region, a pointing device status, a keyboard status, and a current command which are generated from said programmed interaction by said software controller;
(d) waiting in said interaction input component H until said user taking a right interaction input against said running context;
(e) scoring said interaction input by said H;
(f) if said score is higher than a level assigned to a next output of said software simulation automation, then under control of said index component G,
    updating said next output of said software simulation automation invisibly,

forgoing a next sub-process,
canceling a next programmed interaction;
(g) otherwise, visually updating said next output of said software simulation automation;
(h) going back to step (a);
(i) or, reaching a final of said software simulation automation.

62. (Original) The method of claim 60, wherein said interaction input component H, said index component G, and said programmable extension component E are implemented in modular ways so that each component can be bypassed programmatically.

63. (Original) The method of claim 51, further comprising the steps of:
providing a software intelligence device, wherein said software intelligence device comprises,
said model of said software,
said interaction input component H,
said index component G, and
said programmable extension component E;
wherein said software intelligence device connects with said software controller;
wherein said model of said software further comprises
a setup part,
a modeled algorithm engine part, and
an output part;
wherein said setup part is simulated invisibly under control of said index component G;
wherein said modeled algorithm engine part is extended with an input reasoning by said programmable extension component E;
wherein said output part is extended with an output reasoning by said programmable extension component E;
interactively simulating modeled algorithm engine through said interaction input component H by a human user.

64. (Original) The method of claim 51, further comprising the steps of:
providing a software-2, wherein said software-2 comprises an automation loop and an interaction loop;
wherein said automation loop simulates and augments a modeled software dynamic system that runs autonomously;
wherein said interaction loop engages a human user interactively into said automation loop;
wherein said interaction loop couples with said automation loop programmatically.

65. (Original) The method of claim 64, further comprising the steps of:
providing at least one web server;
providing a plurality of same software-2, wherein said plurality of same software-2 are connected to said web server through the Internet;
wherein each of said plurality of same software-2 is run interactively by a user locally;
providing a Master machine, wherein said Master machine is selected from said plurality of software-2;
providing a Master, wherein said Master runs said Master Machine;
providing a group of Learner machines, wherein said group of Learner machines are the rest of said plurality of software-2;
providing a group of Learner, wherein each Learner inside said group of Learners runs each Learner machine inside said group of Learner machines;
providing a HTTP request/response protocol over the Internet, wherein said HTTP request/response protocol is used to control the execution of said group of Learner machines by said Master machine;
synchronizing said state of said Learner machines with said state of said Master machine interactively by said Master through said web server;
communicating among said plurality of software-2 machines interactively through the Internet in real-time.

66. (Original) The method of claim 65, further comprising the steps, communicating between said Master machine and said web server, further comprising the sub-steps of:

(a) said Master machine transiting to a state, wherein said state expects an interaction input from said Master,
(b) said Master machine posting a request including a current sampling k_m to said web server,
(c) said Master machine waiting for a response from said web server
(d) said web server recording said request information including said k_m,
(e) said web server sending said response to said Master, and
(f) releasing said Master machine to continue said simulation,
(g) said Master machine pausing said simulation for a local interaction input from said Master in said local interaction input component H,

(h) said Master machine exchanging message with said group of Learner instantly through the Internet in real-time while said local interaction input component H waits for said interaction input,

(i) said Master making said interaction input to said simulation, and

(j) said Master machine continuing said simulation; and

communicating between anyone of said Learner machines and said web server, further comprising the sub-steps of:

(k) said Learner machine transiting to a state, wherein said state expects an interaction input from said Learner,

(l) said Learner machine posting a request including a current sampling k to said web server,

(m) said Learner machine waiting for a response from said web server,

(n) said web server comparing between said k and said k_m of said Master machine recorded most recently,

if $k < k_m$ then,

web server sending said response to said Learner machine by immediately,

said Learner machine continuing said simulation, and

if $k >= k_m$,then,

web server suspending said response to said Learner machine, and

said Learner machine blocking said simulation,

if $k == k_m$,then,

said Learner machine exchanging messages interactively and instantly through the Internet,

until said Master machine posting its current sampling $k_m > k$ to said web server, then,

said web server sending said response to said Learner machine, and

said Learner machine releasing blocking to continue said simulation;

creating a Master-Machine-Learner interaction loop, wherein said Master-Machine-Learner interaction loop is a dynamic mechanism that couples said Master with said Learners virtually through said software-2 machines over the Internet.

67. (Original) The method of claim 66, wherein said step of exchanging messages instantly through the Internet further comprises the steps of: providing a machine, wherein said machine has said current sampling $k==k_m$; said machine sending a message to said web server, wherein said message is selected from the group consisting of a text, an animation, a script command, a

URL, and combination thereof;
and
web server sending said message to all other machines that have said current sampling $k==k_m$ instantly.

68. (Original) The method of claim 65, wherein said step of communicating among said plurality of software-2 machines interactively through the Internet in real-time, further comprises the steps of:
providing at least one service over the Internet to said group of Learner machines by at least one service provider, wherein said service is a multimedia service delivered in real-time;
controlling said service by said Master machine;
synchronizing said service with said Master machine and said Master.

69. (Original) The method of claim 65, further comprising the steps of:
providing a Master machine, wherein said Master machine is run interactively by a Master;
providing a simulation service over the Internet through real-time communication, wherein said simulation service is published by said Master machine;
providing at least one Learner machine, wherein said Learner machine is run interactively by a Learner;
wherein said Learner machine subscribes to said simulation service over the Internet through real-time communication;
providing a Master-Machine-Learner loop over the Internet, wherein said Master-Machine-Learner loop is a Human-Machine-Human interaction running the same software-2 machines in sync through real-time communication.

70. (Original) The method of claim 69, further comprising the steps of:
providing a book, wherein said book is authored by programming said software controller to drive said software;
projecting at least one of,
(a) a course trajectory running over said software, wherein said course trajectory simulates simulation of the underlying domain knowledge, and
(b) a Lab trajectory running over said software, wherein said Lab trajectory simulates operation of said software;
wherein said simulation of the underlying domain knowledge is programmed in said software;
wherein said operation of said software is programmed in said software;
identifying said model of said software in said software modeling

process;
structuring said model of said software using chapters and keyword indices based on said index component G;
extending a new subject understanding into said model of said software based on said programmable extension component E;
simulating a course content interactively over the Internet in said Master-Machine-Learner loop or locally, wherein said course content is programmed in said course trajectory;
simulating a Lab track interactively over the Internet in said Master-Machine-Learner loop or locally, wherein said Lab track is programmed in said lab trajectory;
organizing teaching-learning activities in a web of software-2 simulation activities.

71. (Currently Amended) A system for modeling and simulating software running
interactively directly or indirectly on at least one digital computer, comprising:
at least one processor;
a display;
a computer readable medium;
a memory, wherein said memory is a part of said computer readable medium coupled to said processor;
an input, wherein said input connects at least one of a pointing device, a keyboard and external interactive devices to said software;
an output, wherein said output connects a block of memory to said display;
a software controller stored on said computer readable medium, wherein said software controller is a programmable agent controlling said software to perform tasks;
a software modeling process implemented on said computer readable medium, wherein said software modeling process models an interaction process executed on said processor between said software and said software controller, further
configured to,

(a) connect said software controller with said software through an input and an output of said software,
(b) control said software by said software controller **automatically and programmatically**, and
(c) identify a model of said software on-line**, wherein said model of said software includes input and output behavior of said software under control of said software controller; wherein said input behavior of said software includes at least one control enabling**

structure; wherein said control enabling structure interacts with said software controller; wherein said output behavior of said software includes at least one display screen region; and

a software simulation process implemented on said computer readable medium, wherein said software simulation process simulates said interaction process executed on said processor between said software and said software controller, further configured to:

(d) connect said software controller with said model of said software through a simulated input and a simulated output of said model of said software,

(e) control said model of said software by said software controller **automatically and programmatically**, and

(f) simulate said interaction process between said software and said software controller without said software presence; and

wherein said software simulation process is a new software that includes said model of said software and said software controller.

72. (previously presented) The system of claim 71, further comprising:

a discrete sampling domain, wherein said discrete sampling domain is a finite integer sequence K driven by said software controller with a current sampling k indicating the most recent sampling.

73. (Original) The system of claim 72, further comprising:

a software dynamic system to represent said software modeling process, wherein said software dynamic system is a discrete system defined over said discrete sampling domain K;

wherein said software simulation process simulates said software dynamic system.

74. (Original) The system of claim 73, further comprising:

an algorithm engine residing in said memory and a cursor engine residing in said memory, wherein said software is factored into said algorithm engine and said cursor engine;

a model of said algorithm engine, wherein said model of said algorithm engine is identified in said software modeling process;

a model of said cursor engine residing in said memory, wherein said model of said cursor engine is identified in said software modeling process;

wherein said model of said algorithm engine and said model of said cursor engine are controlled in parallel by said software controller in said software simulation process; and

wherein an output of said model of said algorithm engine and an

output of said model of said cursor engine are combined as said simulated output of said model of said software in said software simulation process.

75. (Original) The system of claim 74, wherein said cursor engine is modeled,
identified and simulated, further comprising:
a cursor function residing in said memory, wherein said cursor function calculates a cursor output based on a cursor position, a data structure and a plurality of cursor images;
wherein said cursor function models said cursor engine analytically;
a cursor engine modeler residing in said memory, wherein said cursor engine modeler identifies said data structure and said plurality of cursor images while said cursor engine is controlled by said software controller in said software modeling process;
wherein said model of said cursor engine under control of said software controller comprises said cursor function with said identified data structure and said identified cursor images;
wherein said model of said cursor engine simulates said cursor engine in said software simulation process, wherein said software simulation process is executed on said processor, further configured to:
control said model of said cursor engine by said software controller;
calculate said cursor output by said cursor function based on,
said identified data structure,
said identified cursor images, and
said cursor position that is supplied by said software controller;
whereby said cursor engine is simulated by said model of said cursor engine that is controlled by said software controller in said software simulation process without said software presence.

76. (Original) The system of claim 74, wherein said model of said algorithm
engine is modeled and identified in said software modeling process, further comprising:
an initial state of said algorithm engine residing in said memory, wherein said initial state of said algorithm engine is sampled at the first output of said software;
a update function residing in said memory, wherein said update function is a sequence of discrete updates over said discrete

sampling domain;
wherein said update function with said initial state model said algorithm engine;
an algorithm engine modeler residing in said memory, wherein said algorithm engine modeler samples said sequence of discrete updates while said software is controlled by said software controller in said software modeling process;
wherein said model of said algorithm engine identified comprises said sampled initial state and said sampled sequence of discrete updates as a modeled algorithm engine.

77. (Original) The system of claim 76, wherein said algorithm engine modeler is controlled by said software controller and said algorithm engine.

78. (Original) The system of claim 77, wherein said algorithm engine modeler is controlled by two sub-samplers, further comprising:
a direct input and output sampler residing in said memory;
an internal computation sampler residing in said memory; and
wherein said direct input and output sampler and said internal computation sampler run mutually exclusively.

79. (Original) The system of claim 78, wherein said direct input and output sampler controls said algorithm engine modeler to sample at least one discrete update induced directly by a command from said software controller, further comprising means for:
issue said command by said software controller;
wait for $T(\alpha)$, wherein said $\alpha < 1$;
call said direct input and output sampler by said software controller;
trigger said algorithm engine modeler by said direct input and output sampler to sample said discrete update $D(k+\alpha)-D(k)$;
hold said command by said software controller;
advance said current sampling k by one;
model $D(k+\alpha)$ as $D(k+1)$.

80. (Original) The system of claim 79, wherein said direct input and output sampler is callable by said software controller programmatically.

81. (previously presented) The system of claim 78, wherein said internal computation
sampler controls said algorithm engine modeler to sample at least one discrete update that is driven by an internal computation flow inside said algorithm engine, further

comprising:
(a) an internal computation flow residing in said memory, wherein said internal computation flow is executed by said algorithm engine;
(b) at least one anchored point in said internal computation flow, wherein said anchored point is an internal computation unit of said algorithm engine with its computation state transition to be sampled precisely;
(c) a software sensor residing in said memory, wherein said software sensor is attached into said anchor point;
(d) means for setting up said software sensor with a numberOfStates argument, wherein said numberOfStates argument specifies computation state transitions to be observed;
(e) means for driving said algorithm engine to said internal computation unit;
(f) means for calling said software sensor before said internal computation unit is processed;
(g) means for processing said internal computation unit;
(h) means for calling said software sensor after said internal computation unit is processed;
(i) means for subtracting said numberOfStates argument by one;
(j) a logic residing in said memory to check said numberOfStates argument is not zero, wherein if said logic is true, going back to (e); otherwise
(k) means for removing said software sensor.

82. (previously presented) The system of claim 81, wherein said software sensor is an
Event Probe that may perform any programmed function including triggering said sampling by said algorithm engine modeler, further comprising:
an anchored event communicating in said memory, wherein said anchored event is a window message;
wherein said anchored event processing is said anchored point;
means for modifying a callback function address of an internal window message processing;
means for driving said algorithm engine to said internal window message processing;
a dispatch unit residing in said memory, wherein said dispatch unit checks a current message with said anchored event, if a match is found, then,
means for calling a BeforeEvent Probe, wherein said BeforeEvent Probe is said Event Probe, and
means for processing said anchored event processing,

means for calling an AfterEvent Probe, wherein said AfterEvent Probe is said Event Probe, and
means for returning from said internal window message processing;
otherwise, means for processing an internal window message process.

83. (Original) The system of claim 81, wherein said software sensor is an API Probe that may perform any programmed function including triggering said sampling by said algorithm engine modeler, further comprising:
(a) a marker function residing in said memory, wherein said marker function is an application programming call that is modeled as said anchored point within said internal computation flow;
(b) means for modifying a binary execution image of said software to reroute a calling to said marker function to a new function, wherein said new function includes at least one of said API probes and a call to said marker function;
(c) means for driving said algorithm engine to said internal computation flow;
(d) means for proceeding to said marker function;
(e) means for calling a BeforeMarker probe before said marker function is processed, wherein said BeforeMarker is said API Probe;
(f) means for processing said marker function;
(g) means for calling an AfterMarker probe after said marker function is processed, wherein said AfterMarker is said API Probe;
(h) means for subtracting said numberOfStates argument by one;
(i) a logic residing in said memory to check said numberOfStates argument is not zero, wherein if said logic is true, going back to (c); otherwise
(j) means for controlling said algorithm engine to stop said processing of said internal computation flow;
(k) means for removing said BeforeMarker and AfterMarker probes.

84. (Original) The system of claim 81, further comprising:
a controlled target residing in said memory, wherein said controlled target ·is said software in said software modeling process, or said model of said software in said software simulation process;
an input of said controlled target residing in said memory, wherein said input of said controlled target is said input of said

software in said software modeling process, or said simulated input of said model of said software in said software simulation process;
an output of said controlled target residing in said memory, wherein said output of said controlled target is said output of said software in said software modeling process, or said simulated output of said model of said software in said software simulation process.

85. (Original) The system of claim 84, wherein said software controller further comprises,
a work-flow controller residing in said memory, wherein said work-flow controller controls said controlled target to manipulate at least one user interface; and
a work-space controller residing in said memory, wherein said work-space controller drives said controlled target to perform at least one design function.

86. (Original) The system of claim 85, wherein said work-space controller comprises at least one of a coordinate controller residing in said memory and a functional controller residing in said memory.

87. (Original) The system of claim 86, wherein said coordinate controller performs said design function, further comprising:
a list of coordinates residing in said memory, wherein said list of coordinates are used to drive said input of said controlled target.

88. (Original) The system of claim 86, wherein said functional controller performs said design function, further comprising:
at least one modelable function residing in said memory, wherein said modelable function is called with parameters to calculate coordinates;
wherein said coordinates are used to drive said input of said controlled target.

89. (Original) The system of claim 88, further comprising:
at least one software actuator residing in said memory, wherein said software actuator connects said software controller to said controlled target.

90. (Original) The system of claim 89, wherein said software actuator models at least one graphical user interface element of said controlled target.

91. (Original) The system of claim 90, wherein said software actuator comprises a work-flow actuator interfacing to said work-flow controller of said software controller, further comprising means for:
sending a high level command from said work-flow controller to said work-flow actuator, wherein said high level command comprises a tag and an input command;
calculating a valid region based on said tag by said work-flow actuator;
translating said command to at least one primitive action by said work-flow actuator;
if said command is for said pointing device, then
computing a target position by said work-flow actuator based on said valid region,
driving said input of said controlled target to said target position by said work-flow actuator, and
applying said primitive action to said target by said workflow actuator;
else,
applying said primitive action to said valid region by said work-flow actuator.

92. (Original) The system of claim 91, wherein said means for calculating said
valid region based on said tag further comprises means for:
if running in said software modeling process, then
(a) powering said valid region by said software,
(b) identifying said valid region based on said tag on-line,
(c) calculating said valid region, and
(d) saving said valid region with said tag as a part of said model of said software;
otherwise, running in said software simulation process,
(e) simulating said valid region by said model of said software,
(f) retrieving said valid region based on said tag from said model of said software, and
(g) calculating said valid region.

93. (Original) The system of claim 92, wherein said means for identifying said valid region based on said tag in said software modeling process further comprises:
a unique textual name residing in said memory, wherein said unique textual name is said tag;
a unique relation residing in said memory, wherein said unique

relation is coded in a string and assigned as said unique textual name;
means for finding a unique handle from said software based on said tag, wherein said handle is uniquely located by said unique relation that is coded in said tag;
means for identifying said valid region based on said unique handle that is powered by said software.

94. (Original) The system of claim 93, wherein said unique relation is a parent-children-sibling relation that is defined uniquely by said graphical user interface element of said software.

95. (Original) The system of claim 90, wherein said software actuator comprises a work-space actuator interfacing to said work-space controller of said software controller, further comprising means for:
(a) sending a high level command from said work-space controller to said work-space actuator;
(b) calculating a coordinate by said work-space actuator;
(c) translating said high level command to at least one primitive action by said work-space actuator;
(d) driving said input of said controlled target to said coordinate by said work-space actuator;
(e) applying said primitive action to said coordinate by said workspace actuator;
(f) going back to (b) if at least one coordinate remains.

96. (Original) The system of claim 95, wherein said coordinates are calculated by said work-space actuator, further comprising:
a list of coordinates residing in said memory, wherein said list of coordinates is an array of elements with each element having a coordinate (x,y);
wherein said coordinate is retrieved sequentially from said array until the end of said array is reached.

97. (Original) The system of claim 95, wherein said coordinates are calculated
by said work-space actuator, further comprising:
at least one modelable function residing in said memory, wherein said modelable function calculates said coordinate based on at least one parameter that is supplied by said software controller;
wherein said modelable function implemented by said workspace actuator is a mathematical function that is scalable.

98. (Original) The system of claim 90, wherein said software actuator is a system actuator if a system standard graphical user interface element is modeled by said software actuator.

99. (Original) The system of claim 90, wherein said software actuator is a custom actuator if a custom-built graphical user interface element is modeled by said software actuator.

100.(Original) The system of claim 98, wherein said system actuator is reusable universally for a plurality of said software modeling processes and said software simulation processes.

101.(Original) The system of claim 99, wherein said custom actuator is reusable for a plurality of said software controllers for said software modeling process and said software simulation process.

102.(Original) The system of claim 92, wherein said software actuator installs at least one software sensor.

103.(Original) The system of claim 92, further comprising:
a running context of said controlled target residing in said memory, wherein said running context is dynamically reconstructed by said software actuator in said software simulation process;
a structured input and output residing in said memory, wherein said structured input and output structuralizes said input and output of said controlled target based on said running context;
a feedback residing in said memory from said controlled target to said software controller through said structured output of said controlled target;
means for inferring a causal relationship from said running context.

104.(Original) The system of claim 89, further comprising means for:
modularizing said software modeling process and said software simulation process;
replacing said software with said model of said software that is identified in said software modeling process;
switching from said software modeling process to said software simulation process;
preserving said connection between said software controller and said software actuator so that said software controller is kept invariant in said modeling process and said simulation process.

105.(Original) The system of claim 89 wherein said modeled algorithm engine is controlled by said software controller in said software simulation process further comprises:
a Dynamic Update residing in said memory, wherein said Dynamic Update simulates said output of said algorithm engine;
means for reconstructing an output of said modeled algorithm engine, wherein said output of said modeled algorithm engine is updated by said Dynamic Update in sync with said software controller;
at least one simulated sampler residing in said memory, wherein said simulated sampler simulates said sampler used in said software modeling process;
means for controlling said Dynamic Update to update one output of said modeled algorithm engine by said simulated sampler if said sampler samples one discrete update of said algorithm engine in said software modeling process.

106.(Original) The system of claim 105 wherein said simulated sampler is controlled by said software controller through said software actuator.

107.(Original) The system of claim 106 wherein said simulated sampler is a simulated direct input and output sampler that symmetrically simulates said direct input and output sampler used in said software modeling process.

108.(Original) The system of claim 106 wherein said simulated sampler is a simulated internal computation sampler that symmetrically simulates said internal computation sampler used in said software modeling process.

109.(Original) The system of claim 105 wherein said Dynamic Update downloads and streams from an external source to update said output of said modeled algorithm engine.

110.(Original) The system of claim 109 wherein said external source is at least one local file.

111.(Original) The system of claim 109 wherein said external source is at least one Internet server.

112.(Original) The system of claim 107 wherein said software actuator drives said simulated direct input and output sampler, further comprising means for:

modeling a sampling that is induced directly by said command from said software controller;
simulating said sampling by calling said simulated direct input and output sampler.

113.(Original) The system of claim 107 wherein said software actuator simulates said internal computation flow of said algorithm engine, further comprising means for:
modeling an sampling event, wherein said sampling event samples after or before said internal computation unit is processed;
simulating said sampling event by calling said simulated internal computation sampler.

114.(Original) The system of claim 106 further comprising:
a simulated discrete sampling domain residing in said memory, wherein said simulated discrete sampling domain is created from the execution flow of said software controller in said software simulation process;
wherein said simulated discrete sampling domain is identical causally to said discrete sampling domain.

115.(Original) The system of claim 114 further comprising:
a compressing transformation residing in said memory on said simulated discrete sampling domain, wherein said compressing transformation reduces a time duration between two samplings in said software simulation process while causality is preserved;
a stretching transformation residing in said memory on said simulated discrete sampling domain, wherein said stretching transformation extends a time duration between two samplings in said software simulation process while causality is preserved.

116.(Original) The system of claim 73, wherein said software modeling process is a software modeling automation that runs autonomously.

117.(Original) The system of claim 116 wherein said software simulation process is a software simulation automation that runs autonomously, further comprising:
an output of said software simulation automation residing in said memory, wherein said output of said software simulation automation is said output of said model of said software that is manipulable.

118.(Original) The system of claim 117 wherein said software simulation

automation is augmented with additional computation while said software dynamic system is preserved.

119.(Original) The system of claim 118 wherein said software simulation automation is augmented, further comprising:
an interaction input component H residing in said memory, wherein said interaction input component H engages a user to interact with said software simulation automation;
an index component G residing in said memory, wherein said index component G controls visibility of said output of said software simulation automation;
a programmable extension component E residing in said memory, wherein said programmable extension component E extends programmatically said software simulation automation with additional computational process.

120.(Original) The system of claim 119 wherein said interaction input component H is inserted between said output of said software controller and said input of said model of said software, further comprising means for:
(a) updating said software simulation automation at a current sampling k;
(b) executing a programmed interaction by said software controller;
(c) updating a running context, wherein said running context includes a valid region, a pointing device status, a keyboard status, and a current command which are generated from said programmed interaction by said software controller;
(d) transferring execution of said software simulation automation to said H;
(e) waiting for an interaction input from said user;
(f) receiving said interaction input from said user;
(g) comparing said interaction input with said running context;
(h) if said interaction input is matched with said running context, then continuing said execution of said software simulation automation;
(i) otherwise, going back to (e).

121.(Original) The system of claim 119 wherein said interaction input component H is independent of said software simulation automation and said H is applicable universally to any software simulation automation.

122.(Original) The system of claim 119 wherein said interaction input

component H can be plugged or unplugged into said software simulation automation programmatically so that said software simulation automation can run interactively or automatically.

123.(Original) The system of claim 119 wherein said index component G indexes said software simulation automation selectively, further comprising:
(a) at least one index set K_I residing in said memory, wherein said K_I is a subset of said discrete sampling domain K;
(b) means for updating said output of said software simulation automation in said memory that is invisible;
(c) a gated output-mapping function residing in said memory, wherein said gated output-mapping function controls copying from said memory to said display based on said index set K_I;
(d) means for running said software simulation automation;
(e) means for updating a current output of said software simulation automation at a current sampling k in said memory;
(f) if said current sampling $k \in K_I$, then,
  means for copying said current output from said memory to said display by said gated output-mapping function,
  means for continuing said simulation;
(g) otherwise,
  means for advancing said current sampling k by one,
  means for going back to (e) as fast as computationally possible.

124.(Original) The system of claim 123 further comprising:
first textual term residing in said memory, wherein said first textual term is associated with a sampling k_1;
second textual term residing in said memory, wherein said second textual term is associated with a sampling k_2;
at least one pair of descriptive textual terms residing in said memory for searching, wherein one in said descriptive textual terms is said first textual term, and said other in said descriptive textual terms is said second textual term;
means for conducting searches based on said descriptive textual terms to find two samplings k_1 and k_2;
means for running said software simulation automation;
means for updating said current output of said software simulation automation in said memory;
if said current sampling k >= k1 and k < k2, then
  means for copying said output of said software simulation automation from said memory to said display by said gated output-mapping function;
means for selectively running at least one segment of simulation

from k_1 to k_2-1 visually while computationally running said software simulation automation internally.

125.(Original) The system of claim 119 wherein said programmable extension component E extends additional computation into said output of said software simulation automation, further comprising:
a new programmable function, wherein said new programmable function implements a sub-process;
means for updating a current output of said software simulation automation at said current sampling k;
means for extending said new sub-process with said current output as an initial state and a next output of said software simulation automation as a final state;
means for computing at least one sub-state with said new programmable function, wherein said sub-state in said subprocess updates over said output of said software simulation automation;
means for updating said next output of said software simulation automation at said current sampling k+1.

126.(Original) The system of claim 125 wherein said new programmable function is a Microsoft Internet Explorer Browser component, further comprising:
a DHTML page residing in said memory, wherein said DHTML page is loaded by said Microsoft Internet Explorer Browser component;
a Document Object Model (DOM) residing in said memory, wherein said Document Object Model is programmed in said DHTML page;
a URL link residing in said memory, wherein said URL link is encoded in said DHTML page;
a target region residing in said memory, wherein said target region is defined by said Document Object Model;
means for clicking or typing in said target region;
means for navigating to said URL link.

127.(Original) The system of claim 120 wherein said software simulation automation further comprises:
a controlled target residing in said memory, wherein said controlled target is a Microsoft Internet Explorer Browser component;
means for lively running said Internet Explorer Browser component in said software simulation automation;
means for programming web browsing activities by said software

controller;
means for engaging said user with said web browsing activities interactively or automatically.

128.(Original) The system of claim 119 further comprising:
a software amplification device residing in said memory, wherein said software amplification device comprises
said software controller,
said interaction input component H,
said index component G, and
said programmable extension component E;
wherein said software amplification device connects with said model of said software in a top loop;
an augmented output of said model of said software residing in said memory, wherein said augmented output of said model of said software is fed through said software amplification device;
a user, wherein said user senses said augmented output as an input from said software amplification device;
wherein said user is engaged with said software amplification device in a bottom loop;
an output residing in said memory to said software amplification device, wherein said output to said software amplification device is an interactive movement by said user; and
means for interacting said model of said software by said user through said software amplification device.

129.(Original) The system of claim 128 further comprising means for:
(a) running a sub-process in said programmable extension component E to prepare said user to take a right interaction, wherein said sub-process implements at least one programmable function;
(b) executing a programmed interaction, wherein said programmed interaction is performed by said software controller;
(c) updating a running context, wherein said running context includes a valid region, a pointing device status, a keyboard status, and a current command which are generated from said programmed interaction by said software controller;
(d) waiting in said interaction input component H until said user taking a right interaction input against said running context;
(e) scoring said interaction input by said H;
(f) if said score is higher than a level assigned to a next output of said software simulation automation, then under control of said index component G,
updating said next output of said software simulation

automation invisibly,
forgoing a next sub-process,
canceling a next programmed interaction;
(g) otherwise, visually updating said next output of said software simulation automation;
(h) going back to step (a);
(i) or, reaching a final of said software simulation automation.

130.(Original) The system of claim 128 wherein said interaction input component H, said index component G, and said programmable extension component E are implemented in modular ways so that each component can be bypassed programmatically.

131.(Original) The system of claim 119 further comprising:
a software intelligence device residing in said memory, wherein said software intelligence device comprises
said model of said software,
said interaction input component H,
said index component G, and
said programmable extension component E;
wherein said software intelligence device connects with said software controller;
wherein said model of said software further comprises
a setup part residing in said memory,
a modeled algorithm engine part residing in said memory, and
an output part residing in said memory;
wherein said setup part is simulated invisibly under control of said index component G;
wherein said modeled algorithm engine part is extended with an input reasoning by said programmable extension component E;
wherein said output part is extended with an output reasoning by said programmable extension component E;
means for interactively simulating modeled algorithm engine through said interaction input component H by a human user.

132.(Original) The system of claim 119 further comprising:
a software-2, wherein said software-2 comprises an automation loop and an interaction loop;
wherein said automation loop simulates and augments a modeled software dynamic system that runs autonomously;
wherein said interaction loop engages a human user interactively into said automation loop;
wherein said interaction loop couples with said automation loop

programmatically.

133.(Original) The system of claim 132 further comprising:
at least one web server;
a plurality of same software-2 residing in said memory, wherein said plurality of same software-2 are connected to said web server through the Internet;
wherein each of said plurality of same software-2 is run interactively by a user locally;
a Master machine, wherein said Master machine is selected from said plurality of software-2;
a Master, wherein said Master runs said Master Machine;
a group of Learner machines, wherein said group of Learner machines are the rest of said plurality of software-2;
a group of Learner, wherein each Learner inside said group of Learners runs each Learner machine inside said group of Learner machines;
a HTTP request/response protocol, wherein said HTTP request/response protocol is used to control the execution of said group of Learner machines by said Master machine;
means for synchronizing said state of said Learner machines with said state of said Master machine interactively by said Master through said web server;
means for communicating among said plurality of software-2 machines interactively through the Internet in real-time.

134.(Original) The system of claim 133 further comprising means for, communicating between said Master machine and said web server, further comprising means for:
(a) transiting by said Master machine to a state, wherein said state expects an interaction input from said Master,
(b) said Master machine posting a request including a current sampling k to said web server,
m
(c) said Master machine waiting for a response from said web server;
(d) said web server recording said request information including said k , m
(e) said web server sending said response to said Master, and
(f) releasing said Master machine to continue said simulation,
(g) said Master machine pausing said simulation for a local interaction input from said Master in said local interaction

input component H,
(h) said Master machine exchanging message with said group of Learner instantly through the Internet in real-time while said local interaction input component H waits for said interaction input,
(i) said Master making said interaction input to said simulation, and
(j) said Master machine continuing said simulation; and
communicating between anyone of said Learner machines and said web server, further comprising means for:
(k) said Learner machine transiting to a state, wherein said state expects an interaction input from said Learner,
(l) said Learner machine posting a request including a current sampling k to said web server,
(m) said Learner machine waiting for a response from said web server,
(n) said web server comparing between said k and said k_m of said Master machine recorded most recently,
if $k < k_m$ then,
said web server sending said response to said Learner machine immediately,
said Learner machine continuing said simulation, and
if $k >= k_m$, then,
said web server suspending said response to said Learner machine, and
said Learner machine blocking said simulation,
if $k == k_m$,then,
said Learner machine exchanging messages interactively and instantly through the Internet,
until said Master machine posting its current sampling $k > k_m$ to said web server, then,
said web server sending said response to said Learner machine, and
said Learner machine releasing blocking to continue said simulation;
creating a Master-Machine-Learner interaction loop, wherein said Master-Machine-Learner interactive loop is a dynamic mechanism that couples said Master with said Learners virtually through said software-2 machines over the Internet.

135.(Original) The system of claim 134 wherein said exchanging messages instantly through the Internet further comprises:
a machine, wherein said machine has said current sampling $k==k_m$;

means for said machine sending a message to said web server, wherein said message is selected from the group consisting of a text, an animation, a script command, a URL, and combination thereof; and
means for said web server sending said message to all other machines that have said current sampling $k==k_m$ instantly.

136.(Original) The system of claim 133 wherein said communicating among said plurality of software-2 machines interactively through the Internet in real-time, further comprises:
at least one service over the Internet to said group of Learner machines by at least one service provider, wherein said service is a multimedia service delivered in real-time;
means for controlling said service by said Master machine;
means for synchronizing said service with said Master machine and said Master.

137.(Original) The system of claim 133 further comprising:
a Master machine, wherein said Master machine is run interactively by a Master;
a simulation service over the Internet through real-time communication, wherein said simulation service is published by said Master machine;
at least one Learner machine, wherein said Learner machine is run interactively by a Learner;
wherein said Learner machine subscribes to said simulation service over the Internet through real-time communication;
a Master-Machine-Learner loop over the Internet, wherein said Master-Machine-Learner loop is a Human-Machine-Human interaction running the same software-2 machines in sync through real-time communication;

138.(Original) The system of claim 137 further comprising:
a book residing in said memory, wherein said book is authored by programming said software controller to drive said software;
projecting at least one of,
(a) a course trajectory residing in said memory running over said software, wherein said course trajectory simulates simulation of the underlying domain knowledge, and
(b) a Lab trajectory residing in said memory running over said software, wherein said Lab trajectory simulates operation of said software;
wherein said simulation of the underlying domain knowledge is programmed in said software;

wherein said operation of said software is programmed in said software;
means for identifying said model of said software in said software modeling process;
means for structuring said model of said software using chapters and keyword indices based on said index component G;
means for extending a new subject understanding into said model of said software based on said programmable extension component E;
means for simulating a course content interactively over the Internet in said Master-Machine-Learner loop or locally, wherein said course content is programmed in said course trajectory;
means for simulating a Lab track interactively over the Internet in said Master-Machine-Learner loop or locally, wherein said Lab track is programmed in said lab trajectory;
means for organizing teaching-learning activities in a web of software-2 simulation activities.

139.(Currently Amended) A method for modeling and simulating software running interactively directly or indirectly on at least one digital computer, comprising the steps of:
providing a first software, wherein said first software is a binary software that is runnable in the form of EXE or Dynamic Link Libraries (DLL);
providing a second software, wherein said second software is programmed to synthesize at least one of a plurality input actions, wherein said plurality of input actions comprise,
at least one pointing device action,
at least one keyboard action, and**/or**
at least one external input action;
executing said first software under control of said second software **automatically and programmatically** by applying said synthesized input actions to said first software by said second software;
identifying [[capturing]] a model of said first software, [[wherein said model of said first software records an input and output behavior of said first software under control of said second software]]**wherein said model of said first software includes input and output behavior of said first software under control of said second software; wherein said input behavior of said first software includes at least one control enabling structure; wherein said control enabling structure interacts with said second software; wherein said output behavior of said first software includes at least one display screen region**;
controlling said model of said first software by said second software **automatically and programmatically** to simulate said input and

output behavior of said first software under control of said second software;
creating a third software, wherein said third software comprises
said model of said first software and said second software.

140.(Currently Amended) A system for modeling and simulating software running interactively directly or indirectly on at least one digital computer, comprising:
at least one processor;
a computer readable media;
a memory, wherein said memory is a part of said computer readable medium coupled to said processor;
a first software residing in said memory, wherein said first software is a binary software that is runnable in the form of EXE or Dynamic Link Libraries (DLL);
a second software residing in said memory, wherein said second software is programmed to synthesize at least one of a plurality input actions, wherein said plurality of input actions comprise;
at least one pointing device action,
at least one keyboard action, and**/or**,
at least one external input action;
wherein said processor is configured to,
execute said first software under control of said second software **automatically and programmatically** by applying said synthesized input actions to said first software by said second software,
capture a model of said first software, [[wherein said model of said first software records an input and output behavior of said first software under control of said second software]]**wherein said model of said first software includes input and output behavior of said first software under control of said second software; wherein said input behavior of said first software includes at least one control enabling structure; wherein said control enabling structure interacts with said second software; wherein said output behavior of said first software includes at least one display screen region**;
control said model of said first software by said second software **automatically and programmatically** to simulate said input and output behavior of said first software under control of said second software;
a third software residing in said computer readable media,
wherein said third software comprises said model of said first software and said second software.